John von Neumann Institute for Computing

Parallel Exact Inference

Yinglong Xia, Viktor K. Prasanna

published in

Parallel Computing: Architectures, Algorithms and Applications ,
C. Bischof, M. Bücker, P. Gibbon, G.R. Joubert, T. Lippert, B. Mohr, F. Peters (Eds.),
John von Neumann Institute for Computing, Jülich,
NIC Series, Vol. **38**, ISBN 978-3-9810843-4-4, pp. 185-192, 2007.
Reprinted in: Advances in Parallel Computing, Volume **15**,
ISSN 0927-5452, ISBN 978-1-58603-796-3 (IOS Press), 2008.

© 2007 by John von Neumann Institute for Computing
Permission to make digital or hard copies of portions of this work for personal or classroom use is granted provided that the copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise requires prior specific permission by the publisher mentioned above.

http://www.fz-juelich.de/nic-series/volume38

Parallel Exact Inference

Yinglong Xia[1] **and Viktor K. Prasanna**[2]

[1] Computer Science Department
University of Southern California, Los Angeles, U.S.A.
E-mail: yinglonx@usc.edu

[2] Ming Hsieh Department of Electrical Engineering
University of Southern California, Los Angeles, U.S.A.
E-mail: prasanna@usc.edu

In this paper, we present complete message-passing implementation that shows scalable performance while performing exact inference on arbitrary Bayesian networks. Our work is based on a parallel version of the classical technique of converting a Bayesian network to a junction tree before computing inference. We propose a parallel algorithm for constructing potential tables for a junction tree and explore the parallelism of rerooting technique for multiple evidence propagation. Our implementation also uses pointer jumping for parallel inference over the junction tree. For an arbitrary Bayesian network with n vertices using p processors, we show an execution time of $O(nk_m^2 + (wn^2 + wN \log n + r^w wN + r^w N \log N)/p)$, where w is the clique width, r is the number of states of the random variables, k is the maximum node degree in the Bayesian network, k_m is the maximum node degree in the moralized graph and N is the number of cliques in the junction tree. Our implementation is scalable for $1 \leq p \leq n$ for moralization and clique identification, and $1 \leq p \leq N$ for junction tree construction, potential table construction, rerooting and evidence propagation. We have implemented the parallel algorithm using MPI on state-of-the-art clusters and our experiments show scalable performance.

1 Introduction

A full joint probability distribution for any real-world systems can be used for inference. However, such a distribution grows intractably large as the number of variables used to model the system grows. Bayesian networks[2] are used to compactly represent joint probability distributions by exploiting conditional independence relationships. They have found applications in a number of domains, including medical diagnosis, credit assessment, data mining, etc.[6,7]

Inference on a Bayesian network is the computation of the conditional probability of the *query variables*, given a set of *evidence variables* as knowledge. Such knowledge is also known as *belief*. Inference on Bayesian networks can be *exact* or *approximate*. Since exact inference is known to be NP hard[5], several heuristics exist for the same. The most popular exact inference algorithm for multiply connected networks was proposed by Lauritzen and Speigelhalter[2], which converts a given Bayesian network into a junction tree, and then performs exact inference on the junction tree.

Several parallel implementations of exact inference are known[4–6]. However, while some of them only deal with singly connected network; others only consider a part of the independent operations. Shared memory implementations of the various stages of exact inference in parallel exist[3,4], but to the best of our knowledge, this is the first parallel *message passing* implementation of the *entire* process: structure conversion from Bayesian network to junction tree, the construction of potential table, rerooting the tree according to

the evidence provided, and inference calculation with multiple evidence on junction tree. The scalability of our work is demonstrated by experimental results in this paper.

The paper is organized as follows: Section 2 gives a brief background on Bayesian networks and junction trees. We explore the parallel conversion from Bayesian network to junction tree in Section 3. Section 4 discusses parallel exact inference on junction tree. Experimental results are shown in Section 5 and Section 6 concludes the paper.

2 Background

A *Bayesian network* exploits conditional independence to represent a joint distribution more compactly. A Bayesian network is defined as $\mathrm{B} = (\mathbb{G}, \mathbb{P})$ where $\mathbb{G}$ is a *directed acyclic graph* (DAG) and $\mathbb{P}$ is the parameter of the network. The DAG $\mathbb{G}$ is denoted as $\mathbb{G} = (\mathcal{V}, \mathcal{E})$ where $\mathcal{V} = \{A_1, A_2, \ldots, A_n\}$ is the node set and $\mathcal{E}$ is the edge set. Each node A_i represents a random variable. If there is an edge from A_i to A_j i.e. $(A_i, A_j) \in \mathcal{E}$, A_i is called a *parent* of A_j. $pa(A_j)$ denotes the set of all parents of A_j. Given the value of $pa(A_j)$, A_j is conditionally independent of all other preceding variables. The parameter $\mathbb{P}$ represents a group of *conditional probability tables* (CPTs) which are defined as the conditional probability $P(A_j|pa(A_j))$ for each random variable A_j. Given the Bayesian network, a joint distribution $P(\mathcal{V})$ can be rewritten as $P(\mathcal{V}) = P(A_1, A_2, \cdots, A_n) = \prod_{j=1}^{n} Pr(A_j|pa(A_j))$. Fig. 1 (a) shows a sample Bayesian network.

The *evidence variables* in a Bayesian network are the variables that have been instantiated with values e.g. $\mathrm{E} = \{A_{e_1} = a_{e_1}, \cdots, A_{e_c} = a_{e_c}\}$, $e_k \in \{1, 2, \ldots, n\}$. Given the evidence, we can inquire the distribution of any other variables. The variables to be inquired are called *query variables*. The process of *exact inference* involves propagating the evidence throughout the network and then computing the updated probability of the query variables.

It is known that traditional exact inference using Bayes' rule fails for networks with undirected cycles[5]. Most inference methods for networks with undirected cycles convert a network to a cycle-free hypergraph called a *junction tree*. A junction tree is defined as $\mathrm{J} = (\mathbb{T}, \hat{\mathbb{P}})$ where $\mathbb{T}$ represents a tree and $\hat{\mathbb{P}}$ denotes the parameter of the tree. Each vertex $\mathcal{C}_i$ (known as a clique) of $\mathbb{T}$ is a set of random variables. Assuming $\mathcal{C}_i$ and $\mathcal{C}_j$ are adjacent, the *separator* between them is defined as $\mathcal{C}_i \cap \mathcal{C}_j$. All junction trees satisfy the *running intersection property* (RIP)[2]. $\hat{\mathbb{P}}$ is a group of *potential tables* (POTs). The POT of $\mathcal{C}_i$, denoted as $\psi(\mathcal{C}_i)$, can be viewed as the joint distribution of the random variables in $\mathcal{C}_i$. For a clique with w variables, each taking r different values, the number of entries in the POT is r^w. Fig. 1 (d) shows a junction tree.

An arbitrary Bayesian network can be converted to a junction tree by the following steps: *Moralization*, *Triangulation*, *Clique identification*, *Junction tree construction* and *Potential table construction*. Fig. 1 (b) and (c) illustrate a moralized graph and a triangulated graph respectively. Sequential algorithms of the above steps were proposed by Lauritzen et al.[2]. To the best of our knowledge, parallel algorithm for potential table construction has not been addressed in the literature.

In a junction tree, exact inference proceeds as follows: Assuming evidence is $\mathrm{E} = \{A_i = a\}$ and $A_i \in \mathcal{C}_j$, E is *absorbed* at $\mathcal{C}_j$ by instantiating the variable A_i, then renormalizing the remaining constituents of the clique. The effect of the updated $\psi(\mathcal{C}_j)$ is propagated to all other cliques by iteratively setting $\psi^*(\mathcal{C}_x) = \psi(\mathcal{C}_x)\psi^*(\mathcal{S})/\psi(\mathcal{S})$ where $\mathcal{C}_x$ is

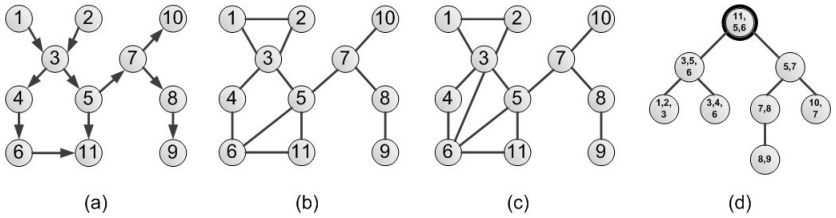


Figure 1. An example of (a) Bayesian network; (b) Moralized graph; (c) Triangulated graph; (d) Junction tree.

the clique to be updated; $\mathcal{S}$ is the separator between $\mathcal{C}_x$ and its neighbour that has been updated; ψ^* denotes the updated POT. After all cliques are updated, the distribution of a query variable $Q \in \mathcal{C}_y$ is obtained by $P(Q) = \sum_{\mathcal{R}} \psi(\mathcal{C}_y)/Z$ where $\mathcal{R} = \{z : z \in \mathcal{C}_y, z \neq Q\}$ and Z is a constant with respect to $\mathcal{C}_y$. This summation sums up all entries with respect to $Q = q$ for all possible q in $\psi(\mathcal{C}_y)$. The details of sequential inference are proposed by Lauritzen et al.[2]. Pennock[5] has proposed a parallel algorithm for exact inference on Bayesian network, which forms the basis of our work.

For the analysis of our work, we have assumed a Concurrent Read Exclusive Write Parallel Random Access Machine (CREW PRAM) model. In implementation, each processor contains a portion of the data, and they interact by passing messages.

3 Parallel Construction of a Junction Tree from a Bayesian Network

Given an arbitrary Bayesian network $\mathrm{B} = (\mathbb{G}, \mathbb{P})$, it can be converted into a junction tree $\mathrm{J} = (\mathbb{T}, \hat{\mathbb{P}})$ by five steps: moralization, triangulation, clique identification, junction tree construction and potential table construction. In this section, we discuss the parallel algorithms used by us in our implementation.

3.1 Structure Conversion

In parallel moralization, additional edges are inserted so that all parents of each node are pairwise connected. The input to our parallel moralization algorithm is a DAG $\mathbb{G} = (\mathcal{V}, \mathcal{E})$. The output is a moralized graph $\mathbb{G}_m = (\mathcal{V}, \mathcal{E}_m)$. The DAG $\mathbb{G}$ is represented as an adjacency list. Each processor is sent a segment of the adjacency array. Assuming node v_i is assigned to processor p_i, p_i generates a set $pa(v_i)$ as the set of v_i's parents and sends $pa(v_i)$ to the processors where v_i's parents are assigned. Each processor works in parallel. Then, every processor p_i receives a set pa_k from its k-th child. If v_i is not connected to a node $v_j \in pa_k$, they are connected so that v_i is moralized.

In triangulation, edges are inserted to $\mathcal{E}_m$ so that in the moralized graph all cycles of size larger than or equal to 4 have chords. A *chord* is defined as an edge connecting two nonadjacent nodes of a cycle. The optimal triangulation minimizes the maximal clique width in the resulting graph. This problem is known to be NP hard[5]. The input to our triangulation algorithm is a moralized graph $\mathbb{G}_m = (\mathcal{V}, \mathcal{E}_m)$. The output is a triangulated graph $\mathbb{G}_t = (\mathcal{V}, \mathcal{E}_t)$. $\mathcal{E}_t$ is the union of the newly added edges and $\mathcal{E}_m$. In order to improve the speed of exact inference in the junction tree, the triangulation method should minimize

the maximum clique width in the resulting graph, which is known to be NP hard[8]. In this step, each processor is in charge of a subset of $\mathbb{G}_m$. The cycles of size equal or larger than 4 are detected and chorded by inserting edges.

In parallel clique identification, the cliques of the junction tree are computed on each processor. The input is a triangulated graph $\mathbb{G}_t = (\mathcal{V}, \mathcal{E}_t)$ and the output is a group of cliques $\{\mathcal{C}_1, \cdots, \mathcal{C}_N\}$. $\mathbb{G}_t$ is also represented as an adjacency array. Each processor obtains a segment of $\mathbb{G}_t$ and forms a candidate clique for each node v_i in parallel. Each processor then uses the details of all the cliques to verify if there exist $\mathcal{C}_i, \mathcal{C}_j$ s.t. $\mathcal{C}_i \subseteq \mathcal{C}_j$. If so, the candidate clique $\mathcal{C}_i$ is removed. Each processor performs the candidate clique verification in parallel. The survived cliques form the output.

Parallel junction tree construction assigns a parent to each clique identified in the previous step. The input to our parallel junction tree construction is a group of cliques $\{\mathcal{C}_1, \cdots, \mathcal{C}_N\}$. The output is junction tree structure $\mathbb{T}$ which is an adjacency array representing the connections between cliques. To satisfy the RIP, we need to compute the union $\mathcal{U}_j = \mathcal{C}_1 \cup \mathcal{C}_2 \cup \cdots \cup \mathcal{C}_{j-1}$ for $\mathcal{C}_j$, $j = 1, \cdots, N$. We use the well-known parallel technique of *pointer-jumping*[1] to perform the computation in $O(\log N)$ time for N cliques. Then, each processor computes in parallel the intersection $\mathcal{I}_j = \mathcal{C}_j \cap \mathcal{U}_j$ and finds $\mathcal{C}_i \supseteq \mathcal{I}_j$ from $\mathcal{C}_1, \cdots, \mathcal{C}_{j-1}$ so that $\mathcal{C}_i$ can be assigned as the parent of $\mathcal{C}_j$.

3.2 Potential Table Construction

Potential table construction initializes a POT for each clique of a junction tree using the CPTs of the Bayesian network from which the junction tree is converted, and then converts the initial POTs to chain set POTs. The input to the parallel potential table construction is the Bayesian network $\mathrm{B} = (\mathbb{G}, \mathbb{P})$ and the structure of the junction tree $\mathbb{T}$. The output is $\hat{\mathbb{P}}$, a group of POTs for the cliques of junction tree $\mathrm{J} = (\mathbb{T}, \hat{\mathbb{P}})$. In parallel potential table construction, each processor is in charge of one or several cliques. The processors work in parallel to identify those nodes A that satisfy $A \cup pa(A) \subseteq \mathcal{C}_i$ for each clique $\mathcal{C}_i$. Then, the CPTs of these identified nodes are multiplied in parallel to produce the initial POT for each clique. Each processor then converts the initial POTs to chain set POTs in parallel. The conversion process is as the same as the inference in junction tree except for the absence of evidence variable. We use a pointer jumping based technique to parallelize the conversion from initial POTs to chain set POTs. The details of the pointer jumping based technique is addressed in Section 4.

3.3 Complexity Analysis

The analysis of the execution time is based on the Concurrent Read Exclusive Write Parallel Random Access Machine (CREW PRAM) model. i.e. concurrent read access to the same data by different processors is allowed, but concurrent write is not.

The execution time of moralization is $O(nk^2/p)$ where n is the number of nodes, k is the maximal number of neighbors in $\mathbb{G}$, and p is the number of processors. Triangulation and clique identification take $O(k_m^2 n + wn^2/p)$ time, where w is the clique width and k_m is the maximal number of neighbors in $\mathbb{G}_m$. The execution time for junction tree construction is $O((wN \log n + wN^2)/p)$ where N is the number of cliques of the junction tree. The potential table construction takes $O(r^w N(w + \log N)/p)$ time where r is the

number of states of a variable. Both moralization and clique identification are scalable over $1 \leq p \leq n$; while junction tree construction and potential table initialization are scalable over $1 \leq p \leq N$.

4 Parallel Inference in Junction Trees

We discuss parallel inference in junction tree in two situations: when the evidence variable is present at the root (see the clique with thick border in Fig. 1 (d)) of the junction tree and when the evidence variable is not present at the root of the junction tree.

When the evidence variable is present at the root, the root absorbs the evidence by instantiating the evidence variable in its POT. Then, the pointer jumping technique is used to propagate the evidence throughout the complete tree: Each clique $\mathcal{C}_i$ sends its POT $\psi(\mathcal{C}_i)$ to its children $ch(\mathcal{C}_i)$ and receives a POT from its parent $pa(\mathcal{C}_i)$. Each clique's POT $\psi(\mathcal{C}_i)$ is updated using $\psi(pa(\mathcal{C}_i))$ independently. As $\mathcal{C}_i$ may have multiple children, the sending of the POT to each child is performed in parallel. Then, we set $ch(\mathcal{C}_i) = ch(ch(\mathcal{C}_i))$, $pa(\mathcal{C}_i) = pa(pa(\mathcal{C}_i))$ and perform the evidence propagation again. Assuming each clique is assigned to a processor, the whole tree is updated in $O(\log D)$ time where D is the depth of the junction tree. Each update requires rewriting a CPT, taking $O(r^w)$ time with one processor.

When the evidence variable is not present at the root, we extend the parallel tree rerooting technique[5] to make the clique that contains the evidence variable as the root of a new junction tree. We compute the depth-first search (DFS) order α of the cliques starting from the clique with evidence variable. Each processor in parallel gets a copy of α and modifies the edge direction if it is inconsistent with α. There is no change in POTs and separators. The execution time of parallel rerooting technique is $O(|\mathrm{E}|Nw/p)$ where $|\mathrm{E}|$ is the number of cliques with evidence. Once the clique containing evidence variable becomes the root, we perform the parallel evidence inference as the same as that in the first situation.

Since the number of cliques containing evidence is bounded by N and $D \leq N$, the execution time of parallel inference algorithm is bounded by $O(r^w N(\log N)/p)$ where $1 \leq p \leq N$.

5 Experiments

5.1 Computing Facilities

We ran our implementations on the DataStar Cluster at the San Diego Supercomputer Center (SDSC)[9] and on the clusters at the USC Center for High-Performance Computing and Communications (HPCC)[10]. The DataStar Cluster at SDSC employs IBM P655 nodes running at 1.5 GHz with 2 GB of memory per processor. This machine uses a Federation interconnect, and has a theoretical peak performance of 15 Tera-Flops. Furthermore, each channel is connected to a GPFS (parallel file system) through a fibre channel. The DataStar Cluster runs Unix with MPICH. IBM Loadleveler was used to submit jobs to batch queue.

The USC HPCC is a Sun Microsystems & Dell Linux cluster. A 2-Gigabit/second low-latency Myrinet network connects most of the nodes. The HPCC nodes used in our experiments were based on Dual Intel Xeon (64-bit) 3.2 GHz with 2 GB memory. The operating system is USCLinux, a customized distribution of the RedHat Enterprise Advanced Server

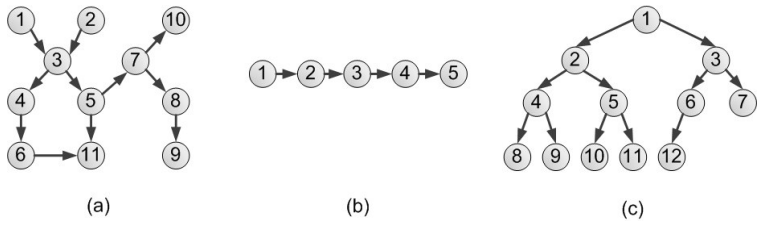


Figure 2. An example of (a) Random Bayesian network; (b) Linear Bayesian network; (c) balanced Bayesian network.

3.0 (RHE3) Linux distribution, with MPICH for communication. The Portable Batch System (PBS) was used for resource allocation and management.

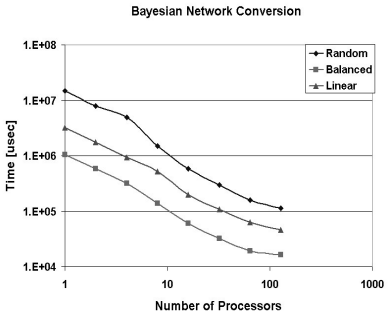
5.2 Input Bayesian Networks

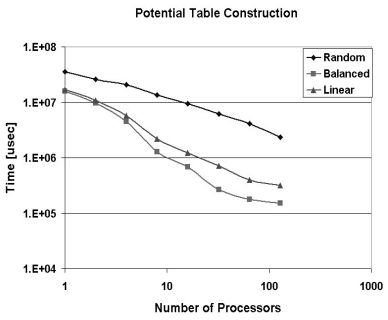
We experimented with three types of Bayesian network as the input: random, linear and balanced. Each Bayesian network has 1024 nodes and each node is binary (r=2). We ran the program with 1, 2, 4, 8, 16, 32, 64 and 128 processors for each of three networks.

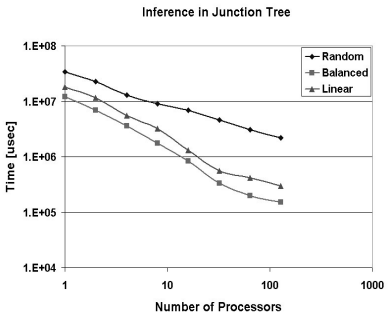
The random Bayesian network (Fig. 2 (a)) was initially generated as a random graph. Representing the initial graph as a adjacency matrix, we modified the matrix to ensure the following constraints: (1) It is an upper-triangular matrix; (2) The graph is connected; (3) The number of 1s in each row and column is limited by a given constant known as *node degree*. These constraints leads to a legal Bayesian network structure. Assuming all nodes are binary, the CPT for a node A was generated as a table of non-negative floating point numbers with 2 rows and $2^{|pa(A)|}$ columns where $|pa(A)|$ is the number of parents of A. In our experiment, the clique width $w = 13$. The linear Bayesian network (Fig. 2 (b)) is a chain: each node except the terminal ones has exactly one incoming edge and one outgoing edge. As $|pa(v)| \leq 1\ \forall v$ and $w = 2$, the CPT size is no larger than 4. The balanced Bayesian network (Fig. 2 (c)) was generated as tree, where the in-degree of each node except the root is 1 and the out-degree for each node except the leaves and last non-leaf node is a constant. So, we have $|pa(v)| \leq 1$ and $w = 2$.

5.3 Experimental Results

In our experiments, we recorded the execution time of parallel Bayesian network conversion, potential table construction and exact inference on junction trees. We added together the above times to obtain the total execution time. The results from DataStar Cluster are shown in Fig. 3 and the results from HPCC in Fig. 4. We are concerned with the scalability of the implementation instead of the speedups. From these figures, we can see that all the individual steps and the total execution time exhibit scalability. Scalability is observed for all three types of Bayesian networks. The execution time for random networks is larger than others because the maximal clique width of the junction tree created from random Bayesian network is larger. The experimental results reflect the scalability of our implementation, confirming that communication does not overshadow computation over a significant range of p.







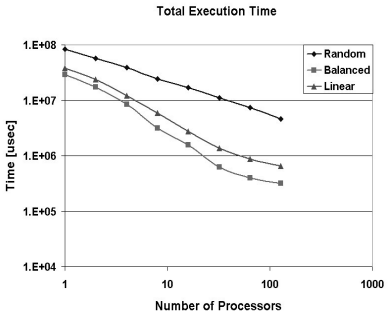
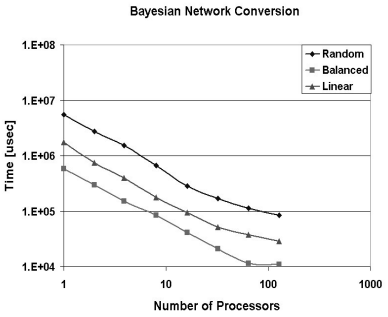
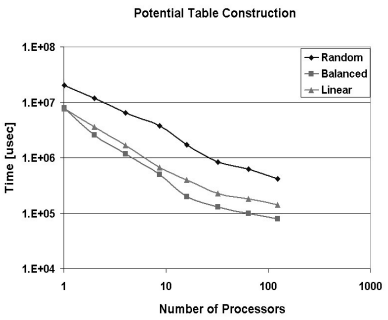
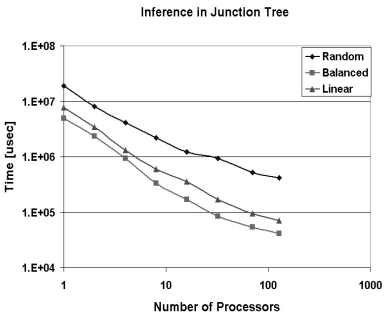


Figure 3. The execution time of exact inference on DataStar.







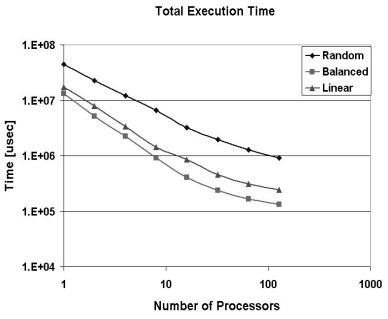


Figure 4. The execution time of exact inference on HPCC.

6 Conclusion

In this paper, we presented our scalable parallel message passing implementation of all the steps involved in exact inference starting from an arbitrary Bayesian network. We also estimate the execution time for each step in this paper. Our implementation was done using MPI and we presented the experimental results from state-of-the-art clusters to show the scalability of our work. For future work, we intend to investigate the parallelization of exact inference at different levels while maintaining scalability.

Acknowledgements

This research was partially supported by the National Science Foundation under grant number CNS-0613376 and utilized the DataStar at the San Diego Supercomputer Center. NSF equipment grant CNS-0454407 is gratefully acknowledged.

References

1. J. JáJá, *An Introduction to Parallel Algorithms*, (Addison-Wesley, Reading, MA, 1992).
2. S. L. Lauritzen and D. J. Spiegelhalter, *Local computation with probabilities and graphical structures and their application to expert systems*, J. Royal Statistical Society B., **50**, 157–224, (1988).
3. V. K. Namasivayam, A. Pathak and V. K. Prasanna, *Scalable parallel implementation of Bayesian network to junction tree conversion for exact inference*, in: Proc. 18th International Symposium on Computer Architecture and High Performance Computing, pp. 167–176, (2006).
4. V. K. Namasivayam and V. K. Prasanna, *Scalable parallel implementation of exact inference in Bayesian networks*, in: Proc. 12th International Conference on Parallel and Distributed Systems, pp. 143–150, (2006).
5. D. Pennock, *Logarithmic time parallel Bayesian inference*, in: Proc. 14th Annual Conference on Uncertainty in Artificial Intelligence, pp. 431–438, (1998).
6. L. Yin, C.-H. Huang, and S. Rajasekaran, *Parallel data mining of Bayesian networks from gene expression data*, in: Poster Book of the 8th International Conference on Research in Computational Molecular Biology, pp. 122–123, (2004).
7. E. Segal, B. Taskar, A. Gasch, N. Friedman and D. Koller, *Rich probabilistic models for gene expression*, in: 9th International Conference on Intelligent Systems for Molecular Biology, pp. 243–252, (2001).
8. T. Klocks, *Treewidth: Computations and Approximations*, Springer-Verlag, Berlin, (1994).
9. `http://www.sdsc.edu/us/resources/datastar/`
10. `http://www.usc.edu/hpcc/`